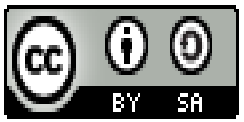
Getting Around to It: Deferred Work in Linux Kernel

Alison Chaiken
alison@she-devel.com
https://github.com/chaiken/SCALE2024

These slides: http://she-devel.com/ChaikenEOS2024.pdf



Categories of Deferred Work

- Tasks delayed due to resource unavailability
- Tasks performed by callbacks in response to an event

Performers of Deferred Work

- Softirqs (bottom halves)
- Kworkers (workqueues)
- Waitqueues
- irq_work

What Happens when Task Deferral Goes Wrong

- Tasks are deferred too long:
  - RCU stalls.
  - Heavy network traffic unacceptably delays applications.
  - Network retransmits.
- Deferred work disrupts latency-sensitive applications:
  - kworkers or ksoftirqd hog cores.
  - kworker watchdog timer fires.

softirqs and workqueues have been inscrutable

http://tinyurl.com/36wh4ssn


softirqs


Softirqs types in order of priority

HI: DMA, PCI

TIMER

NET_TX and NET_RX

BLOCK: assess status of requests

IRQ_POLL: NAPI for storage

TASKLET: crypto, drivers

SCHED: rebalance scheduling domains

HRTIMER

RCU: read-copy-update memory mgmt

http://tinyurl.com/yrhsk9d7

Softirqs: as popular as death

"Softirqs are often a pain to deal with"

"People fight hard through this big softirq lock . . .

"Softirq processing . . . prevents the scheduler to control it . . . heuristics people have added to 'control' this is disgusting"

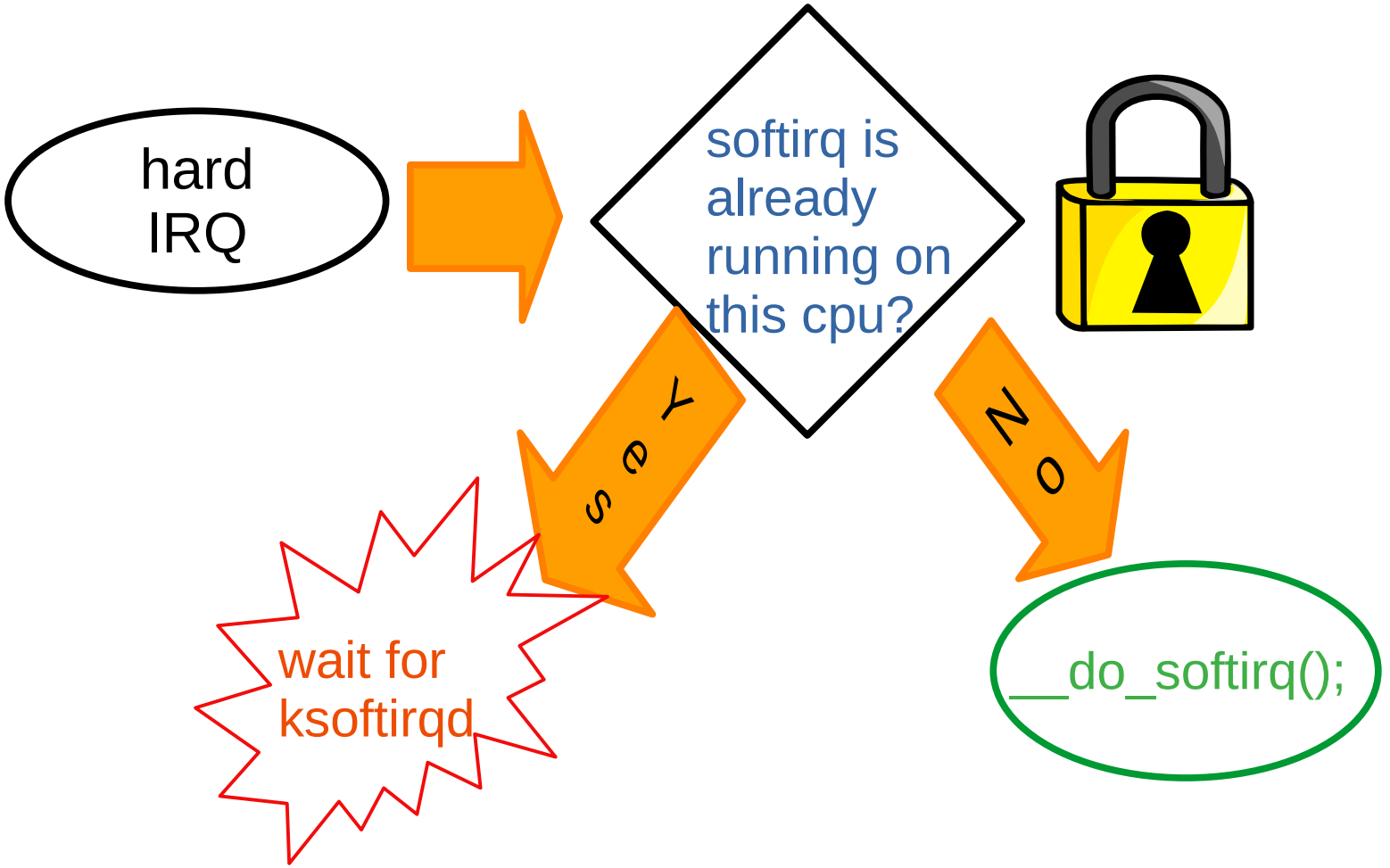
The “disgusting” heuristics

```
/*
 * These limits have been established via experimentation.
 * The two things to balance is latency against fairness -
 * we want to handle softirqs as soon as possible, but they
 * should not be able to lock up the box.
 */

#define MAX_SOFTIRQ_TIME  msecs_to_jiffies(2)
#define MAX_SOFTIRQ_RESTART 10
```

Proposal to add more.

Problem: softirqs do not run concurrently



bcc’s stackcount can make softirqs visible (demo)

```
$ sudo /usr/sbin/stackcount-bpfcc __do_softirq -D 10

Tracing 1 functions for "__do_softirq"... Hit Ctrl-C to end.
 __do_softirq
 run_ksoftirqd
 smpboot_thread_fn                KSOFTIRQD
 kthread
 ret_from_fork
 ret_from_fork_asm
   30

 __do_softirq
 do_softirq
 __local_bh_enable_ip
 iwl_pcie_irq_rx_msix_handler     PIGGYBACK
 irq_thread_fn
 irq_thread
 kthread
 ret_from_fork
 ret_from_fork_asm
   32
```

~~real-time~~ run time latency analyzer (RTLA)

rtla timerlat: Debugging Real-Time Linux Scheduling Latency - Daniel Bristot de Oliveira, Red Hat
331-332 (LEVEL 3)

- Combines the features of ftrace and cyclictest, but easier to interpret.
- Suitable for running in production.
- Captures stacktraces without coredump.

RT Problem: local_bh_disable() defeats PI

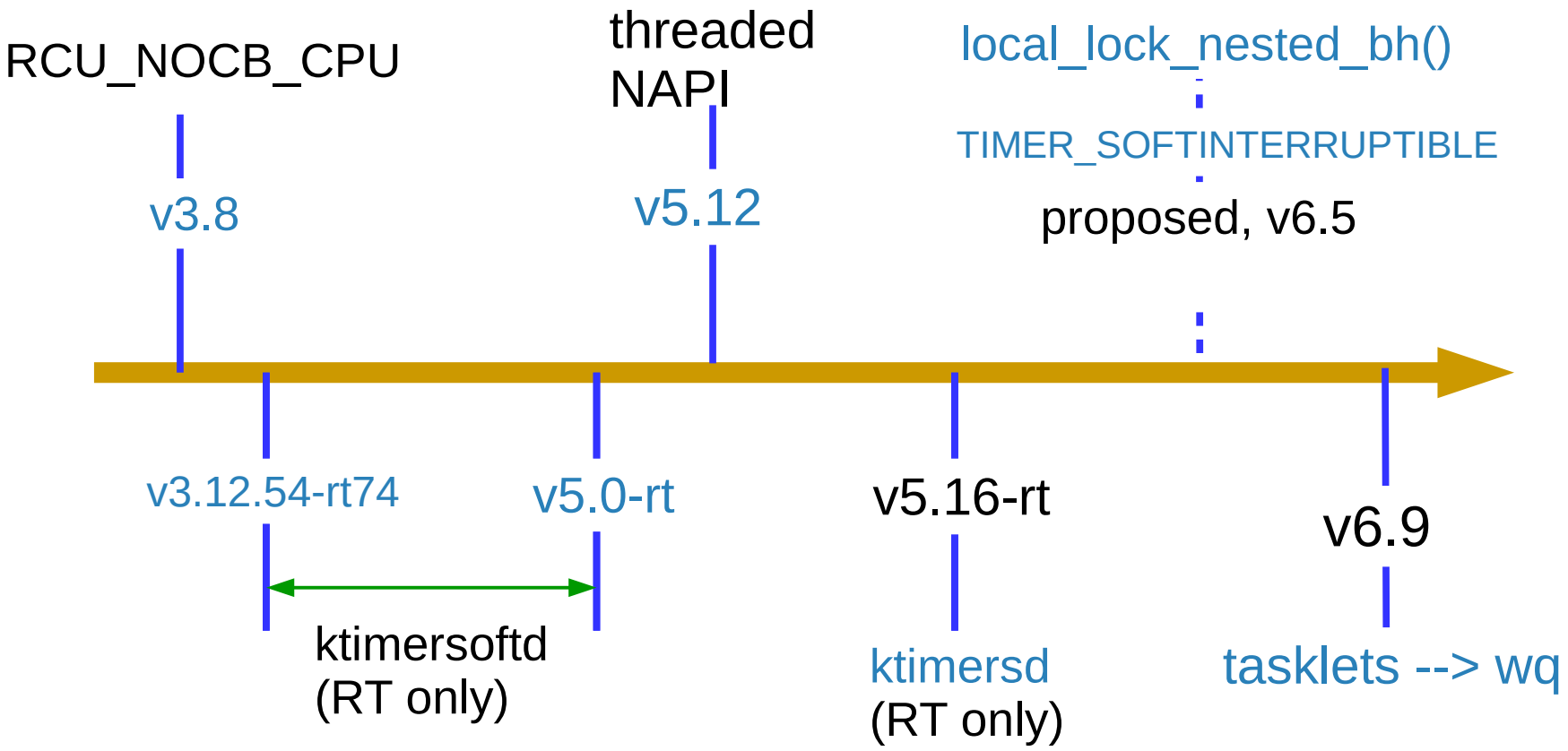
Trace force-threaded interrupts preempted

```
irq/40-eno0-2034 D...2  681 softirq_raise: vec=3 [action=NET_RX]
irq/40-eno0-2034 ..s.2  681 softirq_entry: vec=3 [action=NET_RX]
irq/40-eno0-2034 d.H.3  690 irq_handler_entry: irq=35
irq/40-eno0-2034 dNH33  692 sched_wakeup: irq/35-ahci prio=44 SATA hardirq
irq/40-eno0-2034 d.s23  694 sched_switch: prio=49 R+->irq/35-ahci prio=44
                    Here the BLOCK softirq should run but must wait.
irq/35-ahci-837  d..31  696 sched_pi_setprio: irq/40-eno0 prio 49 -> 44
irq/35-ahci-837  d..21  699 sched_switch: prio=44 D->irq/40-eno0 prio=44
                    The NET_RX softirq is done, so now run BLOCK softirq.
irq/40-eno0-2034 d.s34  715 sched_wakeup: iperf3 prio=120
irq/40-eno0-2034 d..21  736 sched_switch: prio=49 R+->irq/35-ahci prio=44

irq/35-ahci-837  D..13  740 softirq_raise: vec=4 [action=BLOCK]
irq/35-ahci-837  ..s.2  740 softirq_entry: vec=4 [action=BLOCK]
```

S. Siewior, Linux Plumbers 2023 slides, video, LKML

Timeline: incrementally improving softirqs



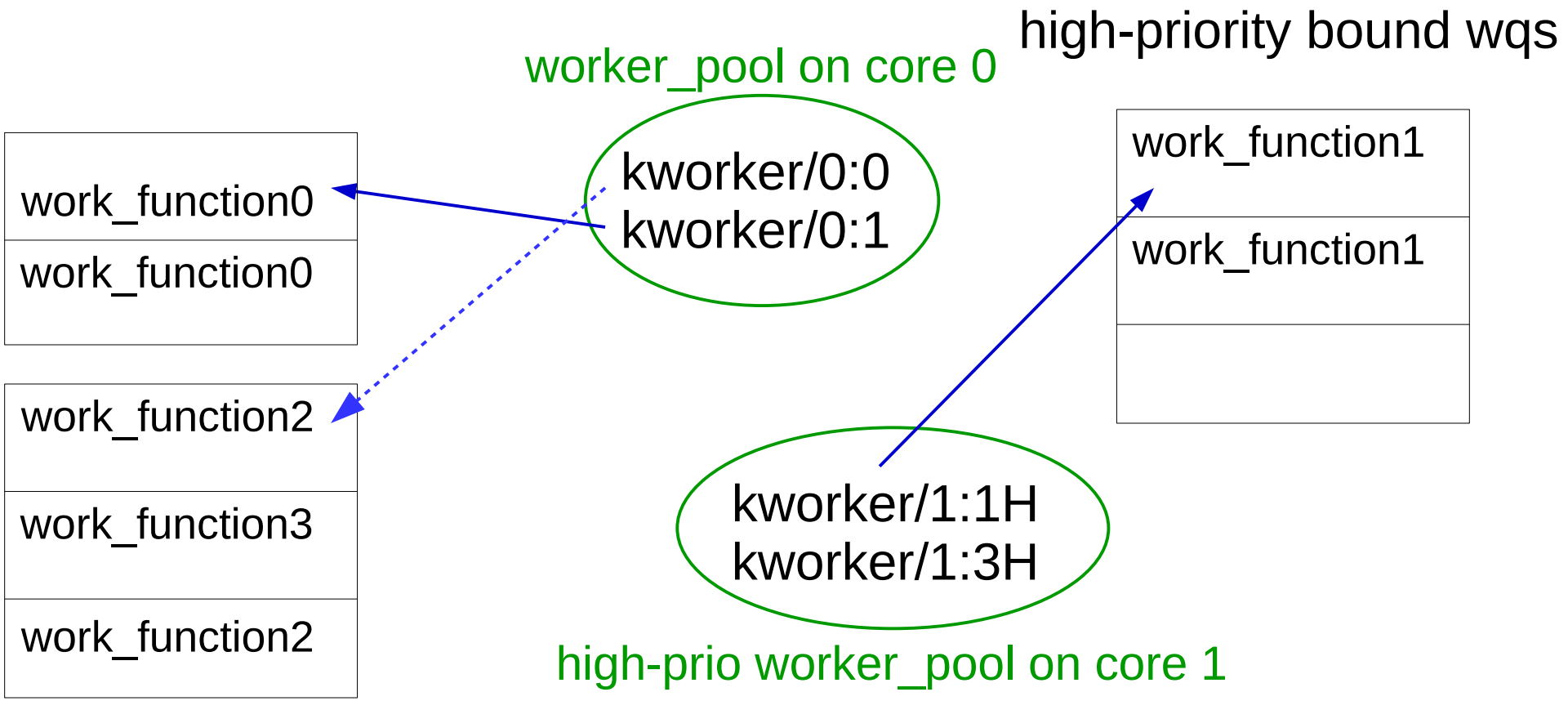
Progress with Softirqs is Slow

credit Estel@deviantart

- ~250 call-sites for local_bh_disable(), the “Big Softirq Lock.”
- RCU, network and timers (RT) softirqs are runnable in kthreads, but with context-switch penalty.
- Tasklets
  move to workqueue API for 6.9.

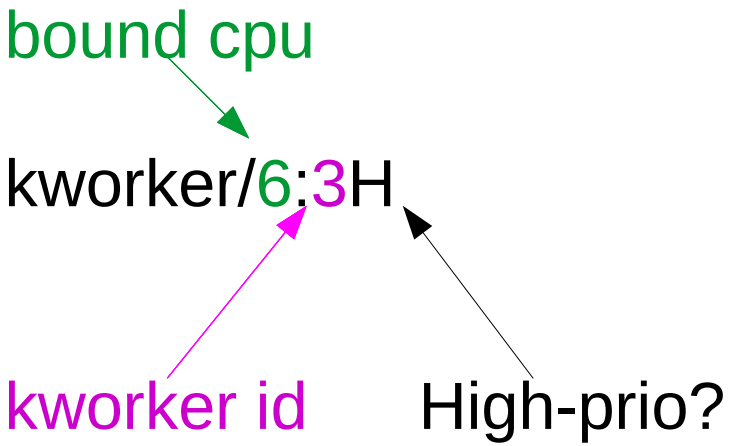

workqueues

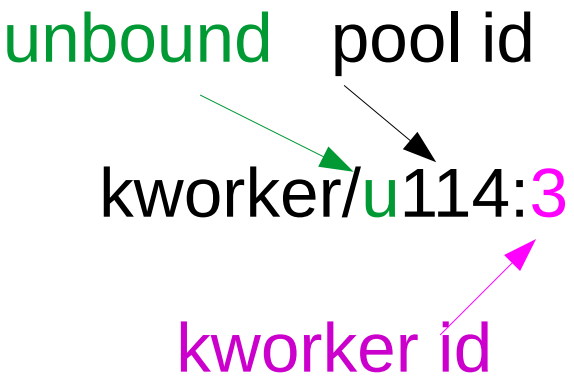

Bound (per-CPU) workqueues



kworker attributes match pools.
A given pool will service diverse workqueues.

kworker naming





Created: at boot
Per-CPU: yes
Fixed # pools: yes
Fixed workers/pool: no
Can migrate: no

Persistent, CPU-intensive
Created: at boot and dynamic
Per-CPU: no
Fixed # pools: no
Fixed workers/pool: no
Can migrate: yes

BUG: workqueue lockup - pool cpus=1 node=0 flags=0x0 nice=0 stuck for 207s!

pool 112: cpus=0-55 flags=0x4 nice=0 hung=0s workers=4 idle: 44535

RT 5.15 kernel

workqueue ixgbe: flags=0xe000a
  pwq 112: cpus=0-55 flags=0x4 nice=0 active=1/1 refcnt=4
    in-flight: 18005:ixgbe_service_task
 workqueue ext4-rsv-conversion: flags=0x2000a
  pwq 112: cpus=0-55 flags=0x4 nice=0 active=1/1 refcnt=14
    in-flight: 53379:ext4_end_io_rsv_work
    inactive: ext4_end_io_rsv_work, ext4_end_io_rsv_work
 workqueue my-deadlocking-driver: flags=0xa000a
   pwq 112: cpus=0-55 flags=0x4 nice=0 active=1/1 refcnt=4
     in-flight: 39998:deadlocking_work_fn

Kernel Thread Pinnability (demo, Github)

$ classify_process_affinity | grep -e ^k

kworker/6:1H-events_highpri: unpinnable
kworker/6:2-mm_percpu_wq: unpinnable
kworker/7:0H-events_highpri: unpinnable
kworker/7:2-mm_percpu_wq: unpinnable
kworker/u16:0-events_unbound: unpinnable
kworker/u17:0-rb_allocator: unpinnable

kcompactd0: pinnable.
kdevtmpfs: pinnable.
khugepaged: pinnable.
khungtaskd: pinnable.
kintegrityd: unpinnable

How it works

struct task->flags & PF_NO_SETAFFINITY

$ cat /proc/93/stat
93 (irq/27-aerdrv) S 2 0 0 0 -1 2129984 0 0 0 0 0 0 0 0 -51 0 1 0 88 0 0
18446744073709551615 0 0 0 0 0 0 0 2147483647 0 0 0 0 17 5 50 1 0 0 0
0 0 0 0 0

Configure workqueues rather than kworkers

https://tinyurl.com/mtvucy4k

https://tinyurl.com/252h7ctt

taskset and chrt manage the wrong thing.

how to set workqueue affinity (demo)

```
[alison@bitscream SCALE2024 (main)]$ sudo ./workqueue-affinity_demo.sh

0.  Demo will not work before v6.7.
Kernel version  6.7-amd64

1. Workqueues which are configurable from sysfs:
$ ls /sys/devices/virtual/workqueue
blkcg_punt_bio  nvme-delete-wq  nvme-wq  raid5wq     scsi_tmf_1  scsi_tmf_3  scsi_tmf_5  scsi_tmf_7  writeback
cpumask         nvme-reset-wq   power    scsi_tmf_0  scsi_tmf_2  scsi_tmf_4  scsi_tmf_6  uevent

2. Consider tunable parameters for nvme-delete-wq:
$ ls /sys/devices/virtual/nvme-delete-wq
affinity_scope  affinity_strict  cpumask  max_active  nice  per_cpu  power  subsystem  uevent

4. Default nice value of unbound nvme-delete-wq workqueue:
$ cat /sys/devices/virtual/workqueue/nvme-delete-wq/nice
0

5. Determine in which workqueue pools nvme-delete-wq runs by default
Workqueue CPU -> pool
=====================
[    workqueue     \     type   CPU  0  1  2  3  4  5  6  7 dfl]
$ drgn tools/workqueue/wq_dump.py | grep nvme-delete-wq
nvme-delete-wq          unbound    66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66 66

6. Set nice to -4
$  echo -4 > /sys/devices/virtual/workqueue/nvme-delete-wq/nice

7. In which workqueue pools does nvme-delete-wq run NOW?
$ drgn tools/workqueue/wq_dump.py | grep nvme-delete-wq

Workqueue CPU -> pool
=====================
[    workqueue     \     type   CPU  0  1  2  3  4  5  6  7 dfl]
nvme-delete-wq          unbound    65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65

9. What are the properties of pool 65?
$ drgn tools/workqueue/wq_dump.py | grep 'pool[65]'
pool[65] ref= 33 nice= -4 idle/workers=  1/  1 cpus=00000015 pod_cpus=00000015

8. What else runs in workqueue pool 65?
$ drgn tools/workqueue/wq_dump.py | grep 65
pool[65] ref= 33 nice= -4 idle/workers=  1/  1 cpus=00000015 pod_cpus=00000015
nvme-delete-wq          unbound    65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65 65
```

Configuration Applies to Work, not Executors

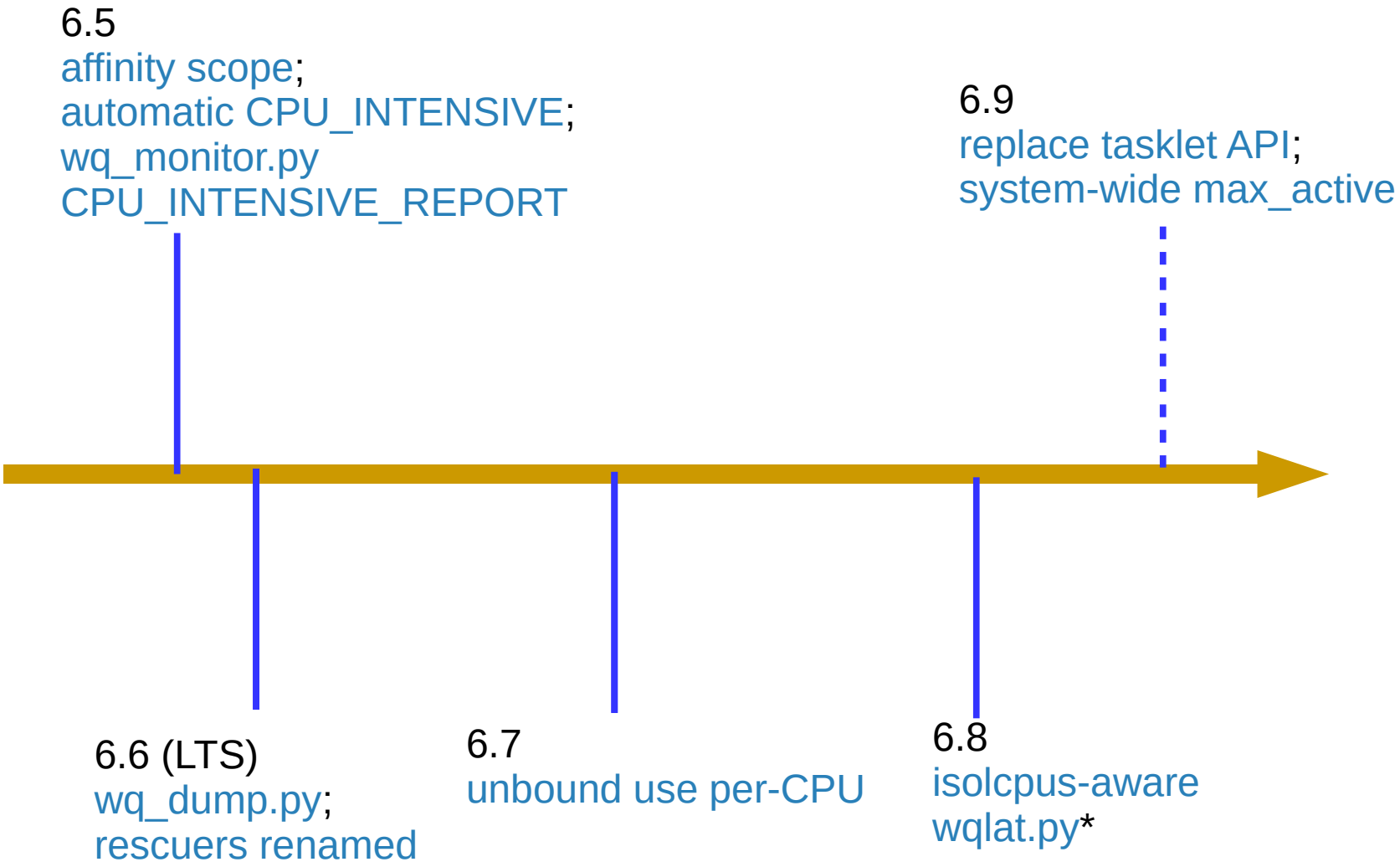
1. *Workqueues* appear in /sys/devices/virtual.

2. *Workqueues* have a “nice” value and cpu affinity.

3. Unbound workqueues can migrate, not kworkers.

4. WQ runtime automatically manages the threads and pools:
   - their number and name are obfuscated.
   - workqueues are NOT exposed to sysfs by default.

Workqueue March of Progress



BH_WQ are Still Softirqs

kernel/**workqueue**.c:

```
/*
 * We don't want to trap softirq for too long. See
 * MAX_SOFTIRQ_TIME and MAX_SOFTIRQ_RESTART in
 * kernel/softirq.c.
 */
#define BH_WORKER_JIFFIES   msecs_to_jiffies(2)
#define BH_WORKER_RESTARTS   10
```

Documentation/core-api/workqueue.rst:
“BH workqueue can be considered a convenience interface to softirq.”

softirqs: high-rate, low-latency, interrupt context, re-entrant (1/CPU)

tasklets: lower-rate softirqs without re-entrancy requirement (1/system)

workqueues: potentially high rate, non-critical latency, process context, sophisticated concurrency mgmt

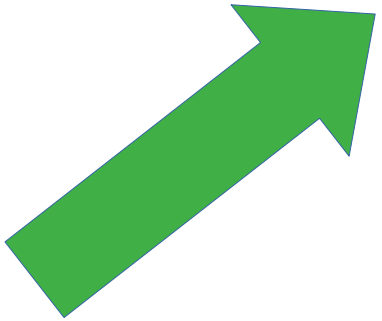
Conclusions

softirqs

Heroic efforts,limited progress.

Improving observability.

workqueues



More configurable.

Improved performance.

Improved observability.

Acknowledgements

Big thanks to Sarah Newman for her suggestions.

References

Best softirq documentation: Robert Love, Linux Kernel Development

Best workqueue documention: in-tree

RTLA paper, slides, website and video

“IRQs: the Hard, the Soft, the Threaded and the Preemptible,” from 2016: video, slides

“Unblocking the softirq lock for PREEMPT_RT” by S. Siewior from 2023: video (starts at 2:16), slides

bpftrace scripts, classify_process_affinity at Github

Comparison by Wei Wang of kthreads, workqueues and softirqs

Helpful kernel configuration

IKHEADERS=y
DEBUG_KERNEL=y
DEBUG_INFO=y
DEBUG_INFO_DWARF_TOOLCHAIN_DEFAULT=y
DEBUG_INFO_BTF=y
DEBUG_INFO_BTF_MODULES=y
PAHOLE_HAS_SPLIT_BTF=y
WQ_CPU_INTENSIVE_REPORT=y
FUNCTION_ERROR_INJECTION=y
BPF_KPROBE_OVERRIDE=y
WQ_WATCHDOG=y
CONFIG_TIMERLAT_TRACER=y
CONFIG_OSNOISE_TRACER=y

Helpful kernel cmdline parameters

General:
nohz_full
isolcpus

Softirqs:
rcu_nocbs
rcu_nocb_poll
rcutree.use_softirq

workqueue.unbound_cpus
workqueue.watchdog_thresh
workqueue.cpu_intensive_thresh_us
workqueue.power_efficient
workqueue.default_affinity_scope

Relevant sysfs attributes

NET_RX softirqs:
$(find /sys -name threaded)
Other softirqs:
/sys/module/kernel/rcu*
/sys/module/srcu*
/sys/module/rcupdate*
/sys/module/rcutree/*

Workqueues:
/sys/module/workqueue/parameters/*
/sys/devices/virtual/workqueue/*

Helpful software

/usr/bin/pahole

drgn + wq_monitor.py or wq_dump.py

bpfcc-tools package or bcc --> wq_lat.py

bpftrace

Understanding Tasklet Softirqs

- Tasklets are event callbacks which:
  - don’t block (no memory allocation, no I/O);
  - have predictable execution time.
- Particular tasklets run once system-wide: minimal locking.
- Heavily used by device drivers.
- Spy on tasklets:

```
$ sudo bpftrace -e 'tracepoint:irq:tasklet_entry { printf("%s\n", ksym(args->func)); }'
```

Yet more workqueue-monitoring tools

- Additional drgn-base kernel tool:
  linux/tools/workqueue/wq_monitor.py

  $ sudo ~/gitsrc/SCALE2024/run-wq_monitor.sh

- New libbpf tool:

  $ sudo python3 ~/gitsrc/bcc/tools/wqlat.py

Network Softirq Sysctl Tunables

dev_weight

The maximum number of packets that kernel can handle on a NAPI interrupt.
Default: 64

dev_weight_rx_bias

Proportion of the configured netdev_budget that is spent on RPS based packet processing during RX softirq cycles.
Default: 1

dev_weight_tx_bias

Scales the maximum number of packets that can be processed during a TX softirq cycle.
Default: 1

New CPU_INTENSIVE_REPORT Feature

Dying USB hub:

workqueue: hub_event hogged CPU for >10000us 4 times, consider switching to WQ_UNBOUND

workqueue: set_brightness_delayed hogged CPU for 10000us 4 times, consider switching to WQ_UNBOUND

6.5 kernel

Intentional Workqueue Throttling

Subject: block: limit request dispatch loop duration
Date: Tue, 5 Apr 2022
From: Shin'ichiro Kawasaki <shinichiro.kawasaki@wdc.com>

When IO requests are made continuously and the target block device handles requests faster than request arrival, the request dispatch loop keeps on repeating to dispatch the arriving requests very long time, more than a minute. Since the loop runs as a workqueue worker task, the very long loop duration triggers workqueue watchdog timeout and BUG [1].

To avoid the very long loop duration, break the loop periodically. When opportunity to dispatch requests still exists, check need_resched(). If need_resched() returns true, the dispatch loop already consumed its time slice, then reschedule the dispatch work and break the loop. With heavy IO load, need_resched() does not return true for 20~30 seconds. To cover such case, check time spent in the dispatch loop with jiffies. If more than 1 second is spent, reschedule the dispatch work and break the loop.

Rescue kworkers

Run when attempt to start more kworkers fails due to ENOMEM.

Each WQ_MEM_RECLAIM has a rescuer kworker which responds to “maydays.”

pre-6.6 called slub_flushwq, inet_frag_wq, etc.

Now called kworker/R*

System Workqueues

Initialized early in boot.

Named “kworker/*events*”.

Used internally by workqueue management.

Also console, tty.

taskset cannot pin workqueues

```
$ sudo taskset -pc 3 8  [kworker/0:0H-events_highpri]
pid 8's current affinity list: 0
taskset: affinity cannot be set due to PF_NO_SETAFFINITY flag
set
taskset: failed to set pid 8's affinity: Invalid argument
```

```
$ sudo taskset -pc 3 913283   [kworker/u17:0-rb_allocator]
pid 913283's current affinity list: 0-7
taskset: affinity cannot be set due to PF_NO_SETAFFINITY flag
set
taskset: failed to set pid 913283's affinity: Invalid argument
```

Unbound Workqueues

Why:

- try to start execution of work items as soon as possible;
- CPU-intensive workloads can be better managed by the system scheduler.

But:

- kworkers can change tasks quickly since there is no context switch.
- kthreads in contrast must wait on the scheduler.

no-threaded-NAPI demo

```
ARM64$ sudo find /sys/ -name threaded
/sys/devices/platform/soc@0/30800000.bus/30be0000.ethernet/net/eth0/threaded
ARM64$ ps ax | grep napi
ARM64$ top
  PID USER    PR  NI    VIRT  RES      SHR S  %CPU  %MEM    TIME+    COMM
1608 debian   20   0   10096   3432   2772  R  11.1     0.2          0:00.04     top

laptop$  netperf -H 10.0.0.2 -t TCP_RR -r 4096 --  -o max_latency,mean_latency

ARM64$  sudo softirqs-bpfcc
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ          TOTAL_usecs
[ . . .]
net_rx              1010045
```

with-threaded-NAPI demo

```
ARM64$  sudo bash -c 'echo 0 >
/sys/devices/platform/soc@0/30800000.bus/30be0000.ethernet/net/eth0/threaded'
ARM64$ ps ax | grep napi
   1038 ?        S      0:00 [napi/eth0-257]
ARM64$ top
PID   USER   PR  NI    VIRT  RES   SHR    S  %CPU  %MEM    TIME+    COMM
1448 root      20   0       0        0        0      S    5.6       0.0      0:01.82   napi/eth0-257


laptop$  netperf -H 10.0.0.2 -t TCP_RR -r 4096 --  -o max_latency,mean_latency

ARM64$  sudo softirqs-bpfcc
Tracing soft irq event time... Hit Ctrl-C to end.
^C
SOFTIRQ          TOTAL_usecs
[ . . .]
net_rx                33925
```

softirqs are
“quicksand code”

https://upload.wikimedia.org/wikipedia/commons/b/ba/Quicksand_warning.jpg